CONTINUOUS MAP ALGORITHMS AND THEIR APPLICATIONS TO DECODE PARALLEL AND SERIAL CODE CONCATENATIONS

S. Benedetto*, D. Divsalar#, G. Montorsi*, F. Pollara#

* Dipartimento di Elettronica, Politecnico di Torino
# Jet Propulsion Laboratory *

Abstract

Concatenated coding schemes with interleavers consist of the combination of two simple *constituent encoders* and an interleaver. The parallel concatenation known as "turbo code" has been shown to yield remarkable coding gains close to theoretical limits, yet admitting a relatively simple iterative decoding technique. The recently proposed serial concatenation of interleaved codes may offer superior performance than turbo codes. In both coding schemes, the core of the iterative decoding structure is a soft-input soft-output (SISO) module. In this paper, we describe the SISO module in a form that continuously updates the MAP probabilities of input and output code symbols, and show how to embed it into iterative decoders for parallel and serially concatenated codes. Results are focused on codes yielding very high coding gain for space applications.

1 Introduction

Concatenated coding scheme have been studied by Forney [1] as a class of codes whose probability of error decreased exponentially at rates less than capacity, while decoding complexity increased only algebraically.

Initially motivated only by theoretical research interests, concatenated codes have since then evolved as a standard for those applications where very high coding gains are needed, such as (deep-)space applications and many others.

The recent proposal of "turbo codes" [2], with their astonishing performance close to the theoretical Shannon capacity limits, have once again shown the great potential of coding schemes formed by two or more codes working in a concurrent way. Turbo codes are *parallel concatenated convolutional codes* (PCCC), in which the information bits are first encoded by a recursive systematic convolutional codes, and then, after passing through an interleaver, are encoded by a second systematic convolutional encoder. The code sequences are formed by the information bits, followed by the parity check bits generated by both encoders. Using the same ingredients, namely convolutional encoders and interleavers, serially concatenated convolutional codes (SCCC) have been shown to yield performance comparable, and in some cases superior, to turbo codes [3].

Both concatenated coding schemes admit suboptimum decoding procedures based on the iterations of the maximum-a-posteriori (MAP) algorithm applied to each constituent code. The purpose of this paper is the description of a module (denoted by SISO) that implements the MAP algorithm in its basic form and the extension of it to the continuous decoding of PCCC and SCCC. As examples of applications, we will show the results obtained by decoding two low-rate codes with very high coding gain, aimed at deep space applications.

2 Iterative Decoding of Parallel and Serial Concatenated Codes

In this section, we show the block diagram of parallel and serially concatenated codes, together with their iterative decoders. It is not within the scope of this paper to describe and analyze the decoding algorithms. For them, the reader is addressed to [2, 4] (for PCCC) and [3] (for SCCC). Rather, we aim at

[1]The research described in this paper was partially carried out at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration, and at the Politecnico of Torino, Italy, under NATO Research Grant CRG 951208

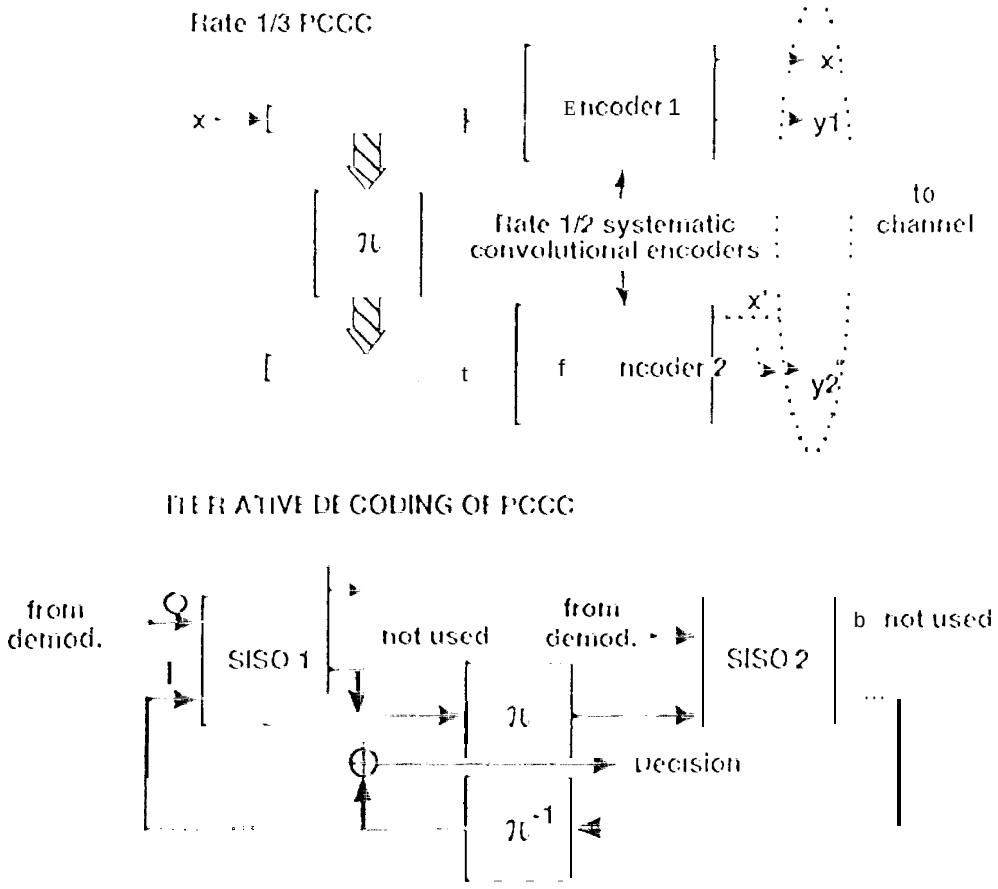

Figure 1: *Block diagram of a parallel concatenated convolutional code*

showing that both iterative decoding algorithms need a particular module, named *soft-input, soft-output* (SISO), which implements operations strictly related to the MAP algorithm, and which will be analyzed in detail in the next section.

2.1 Parallel Concatena ted Codes

The block diagram of a PCCC (the same construction also applies to b lock code.) is shown in Fig. 1. In the figure, a rate 1/3 PCCC is obtained using two rate 1/2 constituent encoders (CCs) and an interleaver. For each input information hit, the codeword sent to the cha n nel is formed by the input bit, followed by the parity check bits ge1 ierated by the two encoders. In Fig. 1 , the block diagram of the ite rative decoder is also sl own. It is based on two modules denoted by "S] S()", one for each encoder, an interleaver and a deinterleaver performing the inverse permutation with respect to the interleaver.

The SISO module is a four-port device, with two inputs and two outputs. A detailed description o f its operations is deferred to the next section. Here, it suffices to say that it accepts as inputs the probability distributio n s of the information arid code symbols labelling the edges of the code trellis, and forms as outputs a1 1 update of these distributions bawd upon the code constraints. It can be noticed, from Fig. "1, that the updated probabilities of the code symbols are never used by the decoding algorithm.

2.2 serially ConcatenatedCodes

The block diagram of a SCCC (the same construction also applies to block codes) is shown in Fig. 2. In the figure, a rate 1/3 SCCC is obtained using as *outer* encoder a rate 1/2 encoder, and as inner encoder a rate 2/3 encoder. An interleaver permutes the output codewords of the outer code before passing them

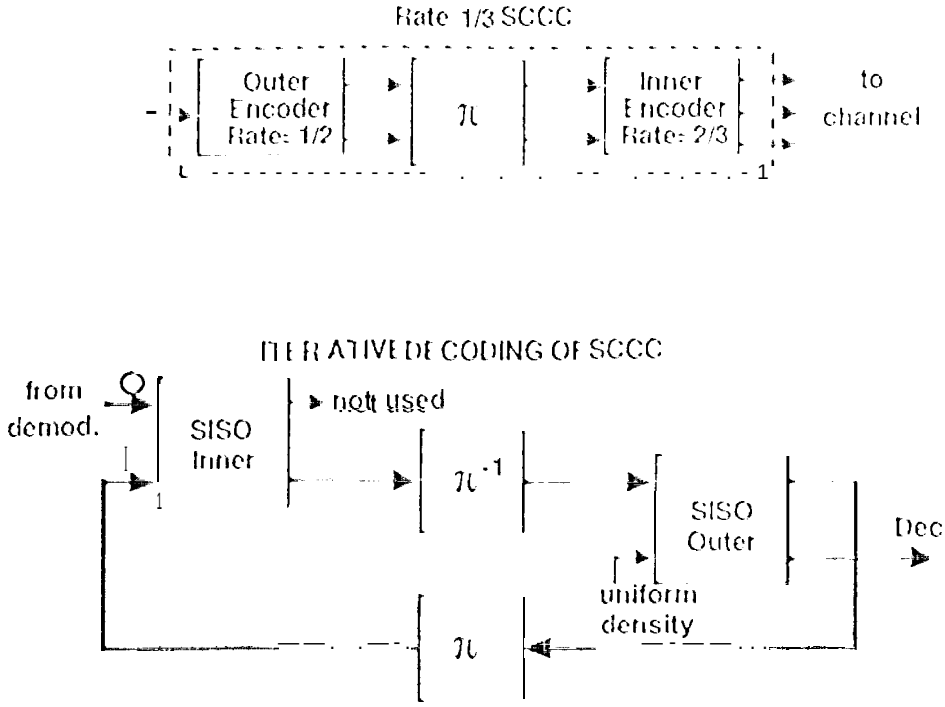

Figure 2: *Serially concatenated convolutional code*

to the inner code. In Fig. 2, the block diagram of the iterative decoder is also shown. It is based on two modules denoted by "S1S()", one for each encoder, an interleaver and a deinterleaver.

The S1SO module is the same as described before. In this case, though, both updated probabilities of input and code symbols are used in the decoding procedure.

2.3 Soft-output algorithms

The SISO module is based on maximum a posteriori (MAP) algorithms. MAP algorithms have been known since the early seventies [5, 6, 7, 8, 9]. The algorithms in [6, 7, 8, 9] perform both forward and backward recursions, and thus require that the whole sequence had been received before starting the decoding operations. As a consequence, they can only be used in block-mode decoding. The memory requirement and computational complexity grow linearly with the sequence length.

The algorithm in [5] requires only a forward recursion, so that it can be used in continuous-mode decoding. However, its memory and computational complexity grows exponentially with the decoding delay. Recently, a MAP symbol-by-symbol decoding algorithm conjugating the positive aspects of previous algorithms, i.e. a fixed delay and linear memory and complexity growth with decoding delay has been proposed in [10].

All previously described algorithms are truly MAP algorithms. To reduce the computational complexity, various forms of suboptimum soft-output algorithms have been proposed. Two approaches have been taken. The first approach tries to modify the Viterbi algorithm. Forney considered "augmented outputs" from the Viterbi algorithm [11]. These augmented outputs include the depth at which all paths are merged, the difference in length between the best and the next-best paths at the point of merging, and a given number of the most likely path sequences. The same concept of augmented output was later generalized for various applications (12, 13, 14, 15, 16]. A different approach to the modification of the Viterbi algorithm was followed in [17], which consists in generating a reliability value for each bit of the hard-output signal: it is called *soft-output Viterbi algorithm* (SOVA). In the binary case, the degradation of SOVA with respect to MAP is small [18]; however, SOVA is not as effective in the non binary case. A comparison of several sub-optimum soft-output algorithms can be found in [19].

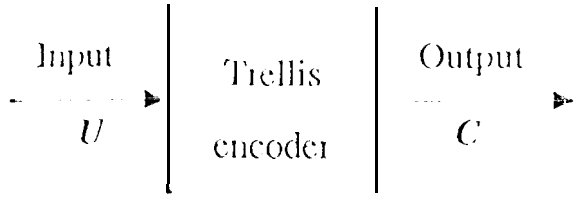

Figure 3: *The trellis encoder*

The second approach consists in revisiting the original symbol MAP decoding algorithms [5, 7], with the aim of simplifying them to a form suitable to implementation [10, 20, 21, 22, 23, 24, 25].

3 The SISO module

3.1 The encoder

The decoding algorithm underlying the behavior of SISO works for codes admitting a trellis representation. It can be a time-invariant or time-varying trellis, and thus the algorithm can be used for both block and convolutional codes. In the following, for simplicity of the exposition, we will refer to the case of *time-invariant convolutional codes*.

In Fig. 3 we show a *trellis encoder*, characterized by the following quantities[1] :

- U = $(U_k)_{k \in K}$ is the sequences of input symbols, defined over a time index set K (finite or infinite) and drawn from the alphabet:

$$\mathcal{U} = \{u_1, \ldots, u_{N_I}\} \ .$$

  To the sequence of input symbols, we associate the sequence of a priori probability distributions:

$$\mathrm{P}(u) = (P_k(u))_{k \in K}$$

  where

$$P_k(u) \stackrel{\Delta}{=} \mathrm{P}[U_k = u]$$

- C = $(C_k)_{k \in K}$ is the sequences of output, or code, symbols, defined over the same time index set K, and drawn from the alphabet:

$$\mathcal{C} = \{c_1, \ldots, c_{N_O}\} \ .$$

  To the sequence of output symbols, we associate the sequence of probability distributions:

$$\mathrm{P}(\mathrm{C}) (P_k(c))_{k \in K}$$

3.2 The trellis section

The dynamics of a time invariant convolutional code is completely specified by a single *trellis section*, which describes the transitions ("edges") between the states of the trellis at time instants k and $k+1$.

A trellis section is characterized by:

- a set of N states $\mathcal{S} = \{s_1, \ldots, s_N\}$. The state of the trellis at time k is $S_k = s$, with $s \in \mathcal{S}$.
- a set of $N \cdot N_I$ edges obtained by the Cartesian product

$$\mathcal{E} = \mathcal{S} \times \mathcal{U} = \{e_1, \ldots, e_{N \cdot N_I}\} \ ,$$

  which represent all possible transitions between the trellis states

[1] In the following, capital letters U, L', <$, E will denote random variables, and lowercase letters u, c, s, e their realizations. The roman letter $\mathrm{P}[A]$ will denote the probability of the event A, whereas the letter $P(a)$ (italic) will denote a function of a. The subscript k will denote a discrete time, defined on the time index set K. Other subscripts, like i, will refer to elements of a finite set. Also, "()" will denote a time sequence, whereas "{}" will denote a finite set of elements.

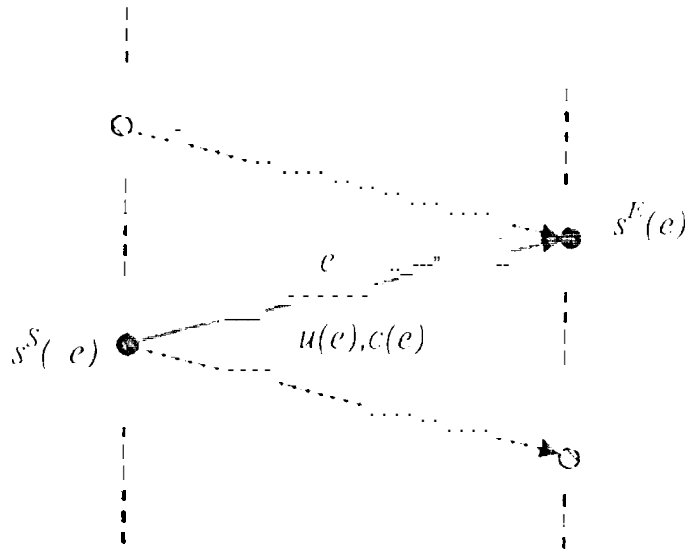

Figure 4: *An edge of the trellis section*

To each edge $e \in \mathcal{E}$ the following functions are associated (see Fig. 4):

- the starting state $s^S(e)$ (the projection of e onto $\mathcal{S}$);
- the ending state $s^E(e)$;
- the input symbol $u(e)$ (the projection of e onto $\mathcal{U}$);
- the output symbol $c(e)$

The relationship between these functions depends on the particular encoder. As an example, in the case of systematic encoders $(s^E(e), c(e))$ also identifies the edge since $u(e)$ is uniquely determined by $c(e)$. In the following, we only assume that the pair $(s^S(e), u(e))$ uniquely identifies the ending state $s^E(e)$; this assumption is always verified, as it is equivalent to say that, given the initial trellis state, there is a one-to one correspondence between input sequences and state sequences, a property required for the code to be uniquely decodable.

3.3 The SISO algorithm

The Soft-Input Soft-Output (SISO) module is a four port device that accepts at the input the sequences of probability distributions:

$$P(c; I) \qquad P(u; I),$$

and outputs the sequences of probability distributions

$$P(c; O) \qquad P(u; O),$$

based on its inputs and on its knowledge of the trellis section (or Code, in general).

We assume first that the time index set K is finite, i.e. $K = \{1, \ldots, n\}$. The algorithm by which the SISO() operates in evaluating the output distributions will be explained in two steps. First, we consider the following algorithm:

- At time k, the output probability distributions are computed as

$$\tilde{P}_k(c; O) = \tilde{H}_c \sum_{e:\, c(e) = c} A_{k-1}[s^S(e)] P_k[u(e); I] P_k[c(e); I] B_k[s^E(e)] \tag{1}$$

$$\tilde{P}_k(u; O) = \tilde{H}_u \sum_{e:\, u(e) = u} A_{k-1}[s^S(e)] P_k[u(e); I] P_k[c(e); I] B_k[s^E(e)] \tag{2}$$

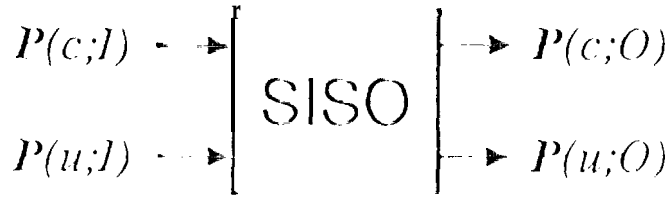

Figure 5: *The Soft-Input Soft-Output (SISO) module*

- The quantities $A_k(\cdot)$ and $B_k(\cdot)$ are obtained through the *forward* and *backward* recursions, respectively, as

$$A_k(s) = \sum_{e:s^E(e)=s} A_{k-1}[s^S(e)]P_k[u(e);I]P_k[c(e);I] \quad , k = 1,\ldots,n \tag{3}$$

$$B_k(s) = \sum_{e:s^S(e)=s} B_{k+1}[s^E(e)]P_k[u(e);I]P_k[c(e);I] \quad , k = n-1,\ldots,0 \ , \tag{4}$$

with initial values:

$$A_0(s) = \begin{cases} 1 & s = s_0 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$B_n(s) = \begin{cases} 1 & s = s_n \\ 0 & \text{otherwise} . \end{cases} \tag{6}$$

The quantities $\tilde{H}_c, \tilde{H}_u$ are normalization constants defined as follows:

$$\tilde{H}_c^{-1} = \sum_c \tilde{P}_k(c;O) = 1$$

$$\tilde{H}_u^{-1} = \sum_u \tilde{P}_k(u;O) = 1$$

From expressions (1) and (2), it is apparent that the quantities $P_k[u(e);I]$ in the first equation and $P_k[c(e);I]$ in the second do *not* depend on e, by definition of the summation indices, and thus can be extracted from the summations. Thus, defining the new quantities

$$P_k(c;O) \triangleq H_c \frac{\tilde{P}_k(c;O)}{P_k(c;I)}$$

$$P_k(u;O) \triangleq H_u \frac{\tilde{P}_k(u;O)}{P_k(u;I)} ,$$

where H_c, H_u are normalization constants such that

$$H_c^{-1} = \sum_c P_k(c;O) = 1$$

$$H_u^{-1} = \sum_u P_k(u;O) = 1 \ ,$$

it can be easily verified that they can be obtained through the expressions

$$P_k(c;O) = H_c\tilde{H}_c \sum_{e:c(e)=c} A_{k-1}[s^S(e)]P_k[u(e);I]B_k[s^E(e)] \tag{7}$$

$$P_k(u;O) = H_u\tilde{H}_u \sum_{e:u(e)=u} A_{k-1}[s^S(e)]P_k[c(e);I]B_k[s^E(e)] \ , \tag{8}$$

where the A's and B's satisfy the same recursions previously introduced in (3).

The new probability distributions $P_k(u;O), P_k(c;O)$ represent a smoothed version of the input distributions $P_k(c;I), P_k(u;I)$, based on the code constraints and obtained using the probability distributions of all symbols of the sequence but the k-th ones $P_k(c;I), P_k(u;I)$. In the literature of "turbo decoding", $P_k(u;O), P_k(c;O)$ would be called *extrinsic informations*. They represent the "added value" of the SISO module to the "a priori" distributions $P_k(u;I), P_k(c;I)$. Basing the SISO algorithm on $P_k(\cdot;O)$ instead than on $\tilde{P}_k(\cdot;O)$ simplifies the block diagrams, and related software and hardware, of the iterative schemes for decoding concatenated codes. For this reason, we will consider as SISO algorithm the one expressed by (7). The SISO module is then represented as in Fig. 5.

Previously proposed algorithms were not in a form suitable to work with a general trellis code. Most of them assumed binary input symbol, some assumed also systematic codes, and none (not even the original BCJR algorithm) could cope with trellis having parallel edges. As it can be noticed from all summations involved in the equations that define the SISO algorithm, we work on trellis edges, rather than on pair of states, and this makes the algorithm completely general, and capable of coping with parallel edges and, also, encoders with rates greater than one, like those encountered in some concatenated schemes.

4 The sliding window soft-input soft-output module (SW-SISO)

As previous description should have made clear, the SISO algorithm requires that the whole sequence had been received before starting the smoothing process. The reason is due to the backward recursion that starts from the (supposed known) final trellis state. As a consequence, its practical application is limited to the case where the duration of the transmission is short (N small), or, for n long, when the received sequence can be segmented into independent consecutive blocks, like for block codes or convolutional codes with trellis termination. It cannot be used for continuous decoding of convolutional codes. This constraint leads to a frame rigidity imposed to the system, and also reduces the overall code rate.

A more flexible decoding strategy is offered by modifying the algorithm in such a way that the SISO module operates on a fixed memory span, and outputs the smoothed probability distributions after a given delay D. We call this new algorithm the *sliding window soft-input soft-output* (SW-SISO) algorithm (and module).

We propose two versions of the SW-SISO, which differ in the way they overcome the problem of initialing the backward recursion without waiting for the entire sequence. From now on, we assume that the time index set K is semi-infinite, i.e. $K = \{1, \cdots, \infty\}$, and that the initial state s_0 is known.

4.1 The first version of the sliding window SISO algorithm (SW-SISO1)

The SW-SISO1 algorithm consists of the following steps:

1. initialize A_0 according to (5).

2. Forward recursion at time k: compute the A_k through the forward recursion (3).

3. Initialization of the backward recursion (time $k > D$):

$$B_k(s) = A_k(s) \quad \forall s \tag{9}$$

4. Backward recursion: it is performed according to (4) from time $k - 1$ back to time $k - D$ as

$$B_{k-1}(s) = \sum_{e:s^S(e)=s} B_k[s^E(e)]P_{k-1}[u(e);I]P_{k-1}[c(e);I], \quad k = k, \ldots, k-D. \tag{10}$$

5. The probability distributions at time $k - D$ are computed as

$$P_{k-D}(c;O) = H_c \tilde{H}_c \sum_{e:c(e)=c} A_{k-D-1}[s^S(e)]P_{k-D}[u(e);I]B_{k-D}[s^E(e)] \tag{11}$$

$$P_{k-D}(u;O) = H_c \tilde{H}_c \sum_{e:u(e)=u} A_{k-D-1}[s^S(e)]P_{k-D}[c(e);I]B_{k-1}[s^E(e)] \tag{12}$$

4.2 The second simplified version of the sliding window SISO algorithm (SW-SISO2)

A further simplification of the sliding window SISO algorithm, that significantly reduces the memory requirements, consists of the following steps:

1. initialize A_0 according to (5).
2. Forward recursion at time k, $k > D$: compute the A_{k-D} through the forward recursion

$$A_{k-D}(s) = \sum_{e:s^S(e)=s} A_{k-D-1}[s^S(e)] P_{k-D}[u(e); I] P_{k-D}[c(e); I] \quad , k > D \, . \tag{13}$$

3. Initialization of the backward recursion (time $k > D$):

$$B_k(s) = \frac{1}{N} \quad \forall s \tag{M}$$

4. Backward recursion (time $k > D$): it is performed according to the previous (i o).
5. The probability distributions at time $k - D$ are computed according to previous (1 1) and (12).

4.3 Memory and computational complexity

4.3.1 Algorithm SW-S SISO1

For a convolutional code with parameters (k_0, n_0), and number of states N, so that $N_I = 2^{k_0}$ and $N_O = 2^{n_0}$, the algorithm SW-SISO1 requires to store $N \times D$ values of A's and $D(N_I + N_O)$ values of the input unconstrained probabilities $P_k(u; I)$, $P_k(c; I)$.

Moreover, to update the A's and B's for each time instant, it needs to perform $2N \cdot N_I$ multiplications and N additions of N_I numbers. To output the set of probability distributions at each time instant, we need a D-times long backward recursion. Thus the computational complexity requires overall:

- $2(D + 1) N \cdot N_I$ multiplications
- $(D + 1) N (N_I - 1)$ additions.

4.3.2 Algorithm SW-SISO2

This simplified version of the sliding window SISO algorithm does not require the storage of the $N \times D$ values of A's, as they are updated with a delay of D steps, As a consequence, only N values of A and $D(N_I + N_O)$ values of the input unconstrained probabilities $P_k(u; I)$, $P_k(c; I)$ need to be stored.

The computational complexity is the same of the previous version of the algorithm. However, since the initialization of the B recursion is less accurate, a larger value of D may be necessary.

5 The additive SISO algorithm (A-SISO)

The sliding-window SISO algorithms solve the problems of continuously updating the probability distributions, without requiring trellis terminations. Their computational complexity, however, is still high when compared to other suboptimal algorithms like SOVA. This is mainly due to the fact that they are *multiplicative* algorithms. In this section, we overcome this drawback by proposing the additive version of the SISO algorithm. The same procedure can obviously applied to its two sliding window versions SW-SISO1 and SW-SISO2.

To convert the previous SISO algorithm from multiplicative to additive form, we exploit the monotonicity of the logarithm function, and use for the quantities $P(u; \cdot)$, $P(c; \cdot)$, A, B their natural logarithms, according to the following definitions:

$$\alpha_k(c; I) \triangleq \log[P_k(c; I)]$$

$$\alpha_k(u; I) \triangleq \log[P_k(u; I)]$$

$$\pi_k(c;O) \stackrel{\Delta}{=} \log[P_k(c;O)]$$

$$\pi_k(u;O) \stackrel{\Delta}{=} \log[P_k(c;O)]$$

$$\alpha_k(s) \stackrel{\Delta}{=} \log[A_k(s)]$$

$$\beta_k(s) \stackrel{\Delta}{=} \log[B_k(s)] .$$

With these definitions, the SISO algorithm defined by equations (7),(8) and (3),(4) becomes the following:

- At time k, the output probability distributions are computed as

$$\pi_k(c;O) = \log\left[\sum_{e:c(e)=c} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \beta_k[s^F(e)]\}\right] + h_c \qquad (15)$$

$$\pi_k(u;O) = \log\left[\sum_{e:u(e)=u} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[c(e);I] + \beta_k[s^F(e)]\}\right] + h_u \qquad (16)$$

where the quantities $\alpha_k(\cdot)$ and $\beta_k(\cdot)$ are obtained through the *forward* and *backward* recursions, respectively, as

$$\alpha_k(s) = \log\left[\sum_{e:s^F(e)=s} \exp\{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \pi_k[c(e);I]\}\right], k = 1,\ldots,n \qquad (17)$$

$$\beta_k(s) = \log\left[\sum_{e:s^F(e)=s} \exp\{\beta_{k+1}[s^S(e)] + \pi_k[u;I] + \pi_k[c(e);I]\}\right], k = n-1,\ldots,0, \qquad (18)$$

with initial values:

$$\alpha_0(s) = \begin{cases} 0 & s = S_0 \\ -\infty & \text{otherwise} \end{cases}$$

$$\beta_n(S_i) = \begin{cases} 0 & s = S_n \\ -\infty & \text{otherwise} \end{cases}$$

The quantities h_c, h_u are normalization constants needed to prevent from an excessive growing of the numerical values of $\alpha's$ and $\beta's$.

The problem in the previous recursions consists in the evaluation of the logarithm of a sum of exponentials like[2]

$$a = \log\left[\sum_i^I \exp\{a_i\}\right] \qquad (19)$$

To evaluate a in (19), we can use two approximations, with increasing accuracy (and complexity). The first approximation is

$$a = \log\left[\sum_i^I \exp\{a_i\}\right] \simeq a_M , \qquad (20)$$

where we have defined

$$a_M \stackrel{\Delta}{=} \max_i a_i , \quad i = 1,\ldots,I$$

This approximation assumes that

$$a_M >> a_i , \quad \forall a_i \neq a_M$$

It is almost optimal for medium-high signal-to noise ratios, and leads to performance degradations of the order of 0.5-0.7 dB for very low signal-to noise ratio.

[2]The notations in this part are modified for simplicity, and do not coincide with previous ones

Using (20), the recursions (17) and (18) become

$$\alpha_k(s) = \max_{e:s^E(e)=s} \{\alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \pi_k[c(e);I]\} \quad k = 1, \ldots, n \tag{21}$$

$$\beta_k(s) = \max_{e:s^E(e)=s} \{\beta_{k+1}[s^S(e)] + \pi_k[u(e);I] + \pi_k[c(e);I]\} \quad k = n-1, \ldots, 0, \tag{22}$$

and the π's of (15) and (16):

$$\pi_k(c;O) = \alpha_{k-1}[s^S(e)] + \pi_k[u(e);I] + \beta_k[s^E(e)] + h_c \tag{23}$$

$$\pi_k(u;O) = \alpha_{k-1}[s^S(e)] + \pi_k[c(e);I] + \beta_k[s^E(e)] + h_u \tag{24}$$

When the accuracy of the previously proposed approximation is not sufficient, we can evaluate a in (19) using the following recursive algorithm (already proposed in [21, 26]):

$$\begin{aligned} a^{(1)} &= a_1 \\ a^{(l)} &= \max(a^{(l-1)}, a_l) + \log[1 + \exp(-|a^{(l-1)} - a_l|)], \quad l = 2, \ldots, L \\ a &= a^{(L)}, \end{aligned}$$

To evaluate a, the algorithm requires to perform $(L-1)$ times two kinds of operations: a comparison between two numbers to find the maximum, and the computation of

$$\log[1 + \exp(-|x|)] \quad |x| \geq 0$$

The second operation can be implemented using a single-entry look-up table up to the desired accuracy (in [21] 8 values were shown to be enough to guarantee almost ideal performance).

The additive form of the SISO algorithm can obviously be applied to both versions of the sliding window SISO algorithms described in the previous section, with straightforward modifications. In the following section, dealing with examples of application, we will use the additive form of the second (simpler) sliding-window algorithm, denoted by additive, sliding-window SISO (ASW-SISO).

6 Applications of the ASW-SISO module

We show in this section examples of applications of the ASW-SISO module embedded into the iterative decoding schemes of PCCCs and SCCCs previously shown in Fig. 1 and 2.

Consider, as a first example, a parallel convolutional concatenated code (turbo code, or PCCC) obtained using as constituent codes two equal rate $1/2$ systematic, recursive, 16-state convolutional encoders with generating matrix

$$G(D) = \left[1, \frac{1 + D + D^3 + D^4}{1 + D^3 + D^4}\right]$$

The interleaver length is $N = 16{,}384$. The overall PCCC forms a very powerful code for possible use in applications requiring reliable operation at very low signal-to-noise ratio, such as deep-space communications systems.

The performance of the continuous iterative decoding algorithm applied to the concatenated code, obtained by simulation using the ASW-SISO and the look-up table algorithms, are shown in Fig. 6, where we plot the bit error probability as a function of the number of iterations of the decoding algorithm for various values of the bit signal-to-noise ratio E_b/N_0. It can be seen that the decoding algorithm converges down to an error probability of 10^{-5} for signal-to-noise ratios of 0.2 dB with nine iterations. Moreover, convergence is guaranteed also at signal-to-noise ratios as low as 0.05 dB, which is as close as 0.55 dB to the Shannon capacity limit.

As a second example, we construct the serial concatenation of two convolutional codes (SCCC) using as outer code the rate $1/2$, 8-state non recursive encoder with generating matrix

$$G(D) = \left[1 + D + D^3, 1 + D\right]$$

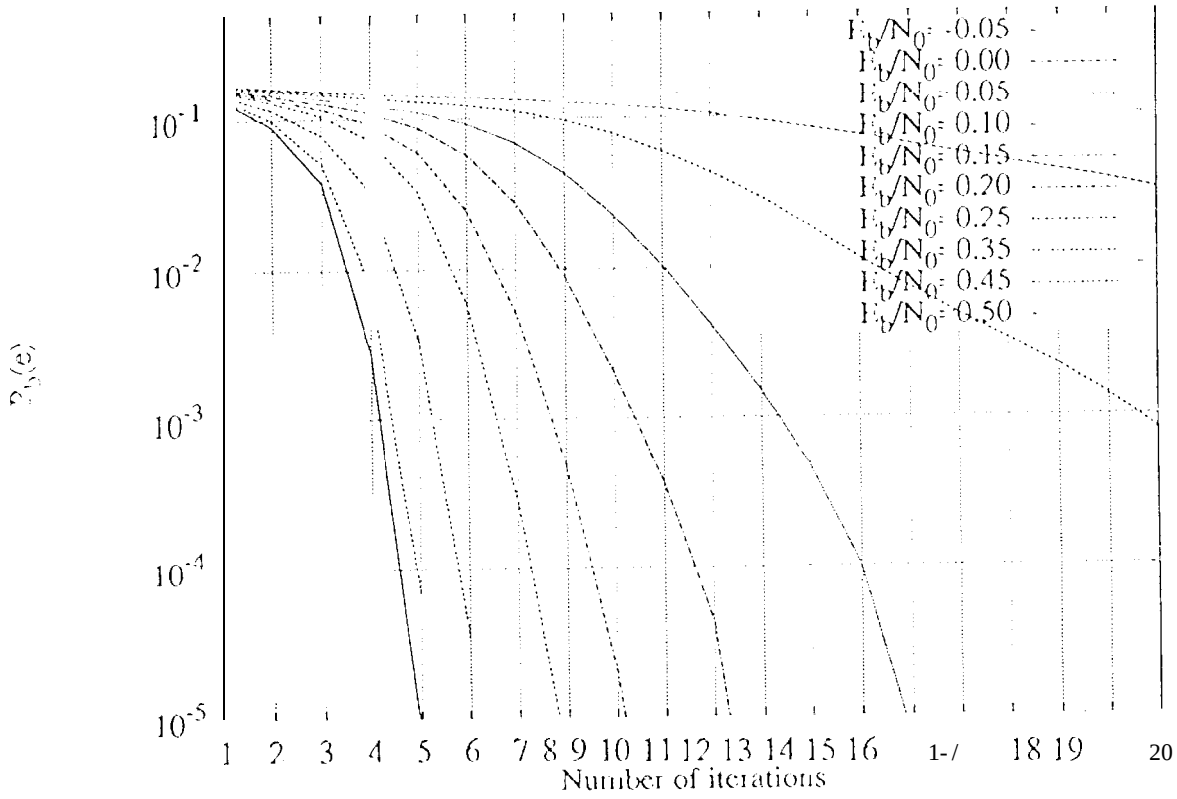

Figure 6: *Convergence of turbo-decoding. Bit error probability versus number of iteration using the ASW-SISO algorithm*

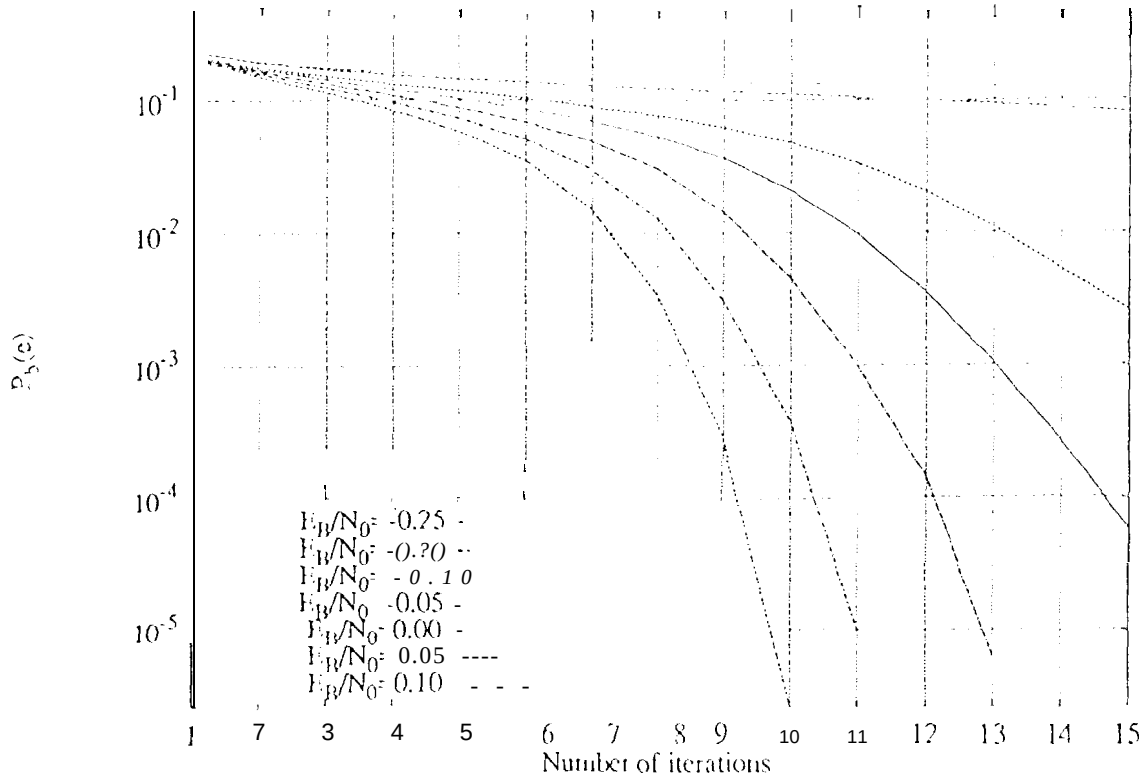

Figure 7: *Convergence of iterative decoding for serial concatenated codes. Bit error probability versus number of iteration using the ASW-SISO algorithm*

and, as inner code, the rate 1/2, 8-state recursive encoder with generating matrix

$$G(D) = \left[1, \frac{1 + D + D^3}{1 + D}\right] .$$

The resulting SCCC has rate 1/4. The interleaver length has been chosen to ensure a decoding delay in terms of input information bits equal to 16,384.

The performance of the concatenated code, obtained by simulation as before, are shown in Fig. 7, where we plot the bit error probability as a function of the number of iterations of the decoding algorithm for various values of the bit signal-to noise ratio E_b/N_0. It can be seen that the decoding algorithm converges down to an error probability of 10^{-5} for signal-to noise ratios of 0.10 dB with 9 iterations. Moreover, convergence is guaranteed also at signal-to noise ratios as low as -0.10 dB, which is as close as 0.71dB to the capacity limit.

As a third, and final, example, we compare the performance of a PCCC and SCCC with the same rate and complexity. The concatenated code rate is 1/3, the CCs are 4-state recursive encoders (rates 1/2 and 1/2 for PCCCs, and rates 1/2 and 2/3 for the SCCCs), and the decoding delays in terms of input bits is equal to 16,384. In Fig. 8, we report the bit error probability versus the signal-to noise ratio for six and nine decoding iterations. As the curves show, the PCCC outperforms the SCCC for high values of the bit error probabilities. Below 10^{-5} (for nine iterations), the SCCC behaves significantly better, and does not present the "floor" behavior typical of PCCCs. In particular, at 10^{-6}, SCCC has an advantage of 0.5 dB with nine iterations.

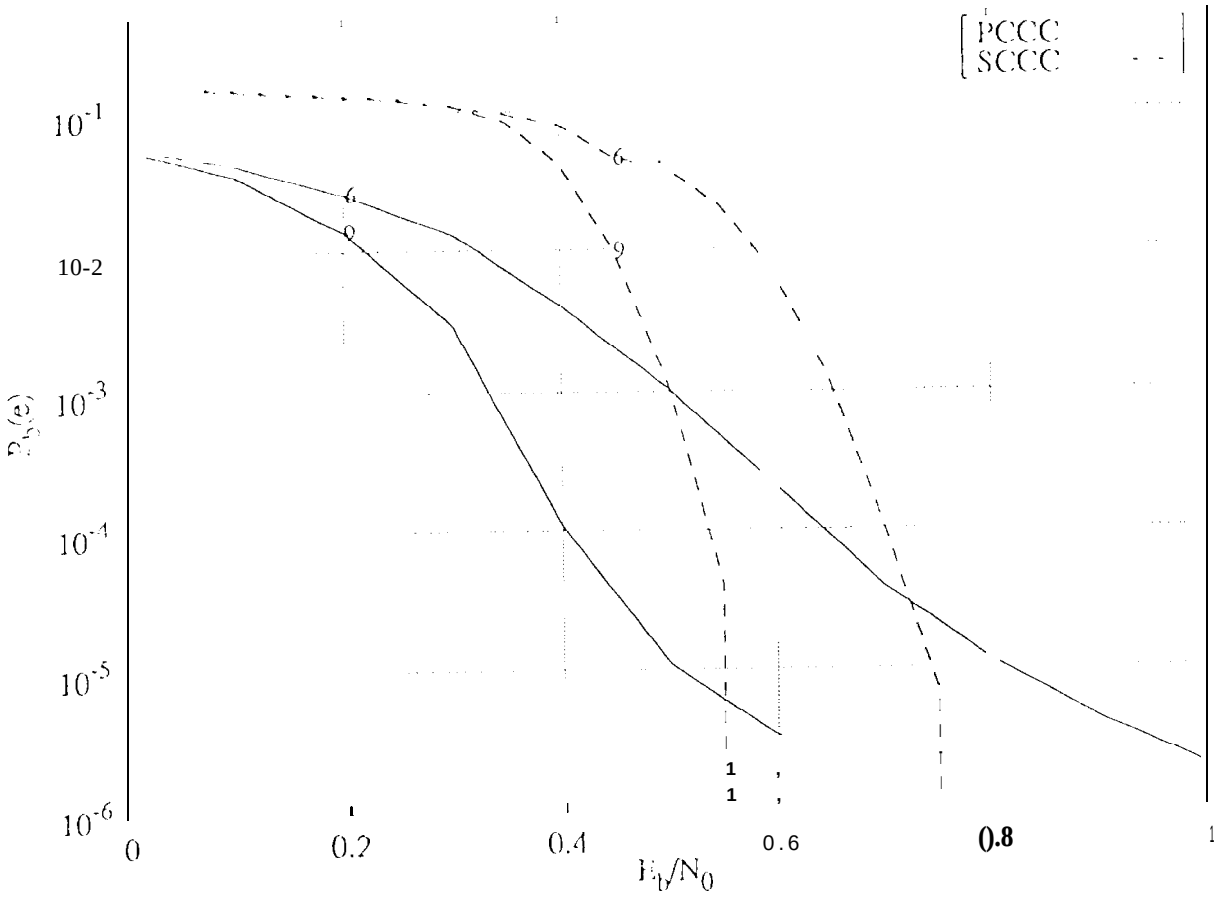

Figure 8: *Comparison* of *two ate 1/3 PCCC and SCCC. The curves refer to six and nine iterations of the decoding algorithm and to an equal input decoding delay of 16384*

7 Conclusions

Algorithms that estimate the maximum-a-posteriori (MAP) probabilities of the information sequence for trellis codes have been synthetically illustrated. Iterative decoding schemes for both parallel and serially concatenated codes need as a key component a module that implements the MAP algorithm. A very general module, called SISO, has been described, which works on the edges of the code trellis section and is able to cope with all possible encoders. While the optimum MAP algorithm is intrinsically block-oriented, we have proposed a sliding-window modification of it that allows continuous decoding of the received stream. Some examples of application have been worked out, concerning very powerful parallel and serially concatenated codes especially suitable for deep-space communication systems.

References

[1] G.D. Forney Jr., *Concatenated Codes*, M.I.T., Cambridge, MA, 1966.

[2] Claude Berrou, Alain Glavieux, and Punya Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes" , in *Proceedings of ICC'93*, Geneve, Switzerland, May 1993, pp. 1064-1070.

[3] Sergio Benedetto and Guido Montorsi, "Iterative decoding of serially concatenated convolutional codes", *Electronics Letters*, July 1996.

[4] D. Divsalar and F. Pollara, "Turbo Codes for PCS Applications", in *Proceedings of ICC'95*, Seattle, Washington, June 1995.

[5] K. Abend and B.D. Fritchman, "Statistical Detection for Communication Channels with Intersymbol Interference", *Proceedings of IEEE*, vol. 58, no. 5, pp. 779-785, May 1970.

[6] R.W. Chang and J.C. Hancock, "On receiver structures for channels having memory", *IEEE Transactions on Information Theory*, vol. IT-12, pp. 463-468, Oct. 1966.

[7] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate", *IEEE Transactions on Information Theory*, pp. 284-287, Mar. 1974.

[8] P.L. McAdam, L. Welch, and C. Weber, "Map bit decoding of convolutional codes", in *Abstract of papers, ISIT'72*, Asilomar, California, Jan. 1972, p. 91.

[9] C.R. Hartmann and L.D. Rudolph, "An optimum symbol-by-symbol decoding rule for linear codes", *IEEE Transactions on Information Theory*, vol. IT-22, pp. 514-517, Sept. 1976.

[10] Branka Vucetic and Yunxin Li, "A Survey of Soft-Output Algorithms", in *Proceedings of ISITA'94*, Sydney, Australia, Nov. 1994, pp. 863-867.

[11] G.D. Forney Jr., "The Viterbi Algorithm", *IEEE Transactions on Information Theory*, vol. IT-61, no. 3, pp. 268-278, Mar. 1973.

[12] H. Yamamoto and K. Itoh, "Viterbi Decoding Algorithm for Convolutional Codes with Repeat Request" , *IEEE Transactions on Information Theory*, vol. IT-Y(i, no. 5, pp. 540-547, Sept. 1980.

[13] T. Hashimoto, "A List-Type Reduced-Constraint Generalization of the Viterbi Algorithm", *IEEE Transactions on Information Theory*, vol. IT-33, no. 6, pp. 866-876, Nov. 1987.

[14] R.H. Deng and D.J. Costello, "High Rate Concatenated Coding Systems Using Bandwidth Efficient Trellis Inner Codes", *IEEE Transactions on Communications*, vol. COM-37, no. 5, pp. 420-427, May 1989.

[15] N. Seshadri and C-E.W. Sundberg, "Generalized Viterbi Algorithms for Error Detection with Convolutional Codes" , in *Proceedings of GLOBECOM'89*, Dallas, Texas, Nov. 1989, vol. 3, pp. 43.3.1-43.3.5.

[16] "T. Schaub and J.W. Modestino, "An Erasure Declaring Viterbi Decoder and its Applications to Concatenated Coding Systems", in *Proc. of ICC'86*, Toronto, Canada, June 1986, pp. 1612-1616.

[17] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications", in *Proceedings of GLOBECOM'89*, Dallas, Texas, Nov. 1989, pp. 47.1.1-47.1.7.

[18] P. Hoeher, "Tcm on frequency-selective fading channels: a comparison of soft-output probabilistic equalizers", in *Proceedings of GLOBECOM'90*, San Diego, California, Dec. 1990, pp. 401.4.1-401.4.6

[19] Ulf Hansson, "Thoretical Treatment of ML Sequence Detection with a Concatenated Receiver", Tech. Rep. 185L, School of Electrical and computer engineering, Chalmers University of technology Göteborg,, Sweden, 1994.

[20] S.S. Pietrobon and A .S. Barbulescu, "A simplification of the modified ba hl algorithm for systematic convolutionalcodes", in *Proceedings of ISITA'94*, Sydr ney, Aus tralia, Nov. 1994, pp. 1073- 1077

[21] Patrick Robertson, Emmanuelle Villebrun, and Peter Hoeher, "A Comparison of Optimal arid Sub-Optimal MAP Decoding Algorithms Operating in the L og Domain", in *Proceedings of IC'D' '95*, Seattle, Washington, June 1995, pp. 1009 1013.

[22] Peter Jung, "Novel Low Complexity Decoder for Turbo Codes", *Electronic Letters*, vol. 31, no. ?, pi), 86-87, Jan. 1995.

[23] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Algorithm for continuous decoding of turbo codes", *Electronics Letters*, vol. 32, no. 4, pp. 314- 315, Feb. 1996.

[24] L. Papke, "Improved decoding with the sova in a parallel concatenated (turbo code) scheme", in *Proceedings of ICC'96*, Dallas, Texas, June 1996.

[25] Sergio Benedetto, Dariush Divsalar, Guido Montorsi, and Fabrizio Pollara, "Soft-output decoding algorithms for continuous decoding of parallel concatenated convolutional codes" in *Proceedings of ICC'96*, Dallas, Texas, June 1996.

[26] Sergio Benedetto, Guido Montorsi, Dariush Divsalar, and Fabrizio Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes", *JPL TDA Progress Report 42-124*, Feb. 1996.